UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

Aplicación de técnicas de aprendizaje automático para la detección de emociones en personas con TEA

Eduardo Hernanz Rodríguez
Tutor: Juan Carlos Torrado Vidal
Ponente: Germán Montoro Manrique

Marzo 2018

Aplicación de técnicas de aprendizaje automático para la detección de emociones en personas con TEA

AUTOR: Eduardo Hernanz Rodríguez
TUTOR: Juan Carlos Torrado Vidal

AmIlab
Dpto. de Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Marzo de 2018

Resumen

El objetivo del proyecto es llegar a una conclusión de optimización de detección de estrés en niños con TEA por medio de clasificación automática. La aplicación **Taimun-Watch** desarrollada por el laboratorio Ambient Intelligence Laboratory (AmILab) de la EPS contiene un método de detección de estrés por medio de sobrepasar un umbral fijo, el objetivo final de este proyecto es el de encontrar el clasificador más preciso, y con menor porcentaje de error que sea capaz de identificar las situaciones de estrés de los niños utilizando los datos que recoge el reloj inteligente (smartwatch) que llevan en la muñeca.

Los clasificadores están actualmente en auge y existe mucha variedad de modelos, cada uno realiza la predicción necesaria por medio de distintos métodos, teniendo cada uno de ellos ventaja sobre el resto en determinadas situaciones, y desventaja en otras. Para la detección de estrés hay un o unos clasificadores que son más precisos que el resto, y en este proyecto se realiza el estudio que encuentra dicho/s clasificador/es.

Para cumplir dicho objetivo, se implementará un código capaz de generar distintos clasificadores y utilizando un conjunto de datos proporcionado por expertos y datos reales recogidos directamente de los smartwatches de niños con TEA, se entrenará un modelo de un conjunto de clasificadores distintos. Tras realizar el entrenamiento de los datos existentes se aplicará el uso de la validación cruzada, una técnica de validación que permite sacar resultados de los clasificadores para que sean evaluados, utilizando particiones de un mismo conjunto de datos como modelos entrenados, y otras partes como testeo del clasificador.

Al resultado de este proyecto le seguirá la implementación del modulo, el cual será posteriormente integrado en la aplicación por un desarrollador de esta, en ella se utilizará el modelo elegido, previamente entrenado, para clasificar los nuevos datos obtenidos a tiempo real por los smartwatches. Para que, en caso de predecir estrés, el smartwatch active el proceso de ayuda del niño.

Palabras clave

Aprendizaje automático, autismo, clasificación, modelo, sistema, entrenamiento, validación, optimización.

Abstract

The objective of the project is to use machine learning to optimize stress detection in children with ASD. The **Taimun-Watch** app, developed by the Ambient Intelligence Laboratory (AmILab) of the EPS, has a method of stress detection surpassing a fixed threshold, the final target of this project is to find the most accurate classifier, which can identify stress situations of children using the data from the smartwatch that they wear.

These days, there is a boom of classifiers and there are a lot of different models, each one performs the necessary prediction using different methods, each of them have advantages and disadvantages over the others. For stress detection there are classifiers which are more optimal than the others, in this project we will find the classifier with the highest performance.

To achieve this goal, I will code a program that generates different classifiers using a dataset that contains real data collected directly from the smartwatches of children with ASD. I will train different classifiers using the dataset and I will use cross-validation to find the most optimal classifier.

After obtaining the results of the study, the most optimal classifier will be implemented inside the application improving the actual stress detection system.

Keywords

Machine learning, autism, classification, model, system, training, validation, test.

Agradecimientos

Me gustaría dar las gracias a todos los que me han ayudado y apoyado a lo largo del grado, en especial, durante los años más duros y durante el desarrollo de este proyecto.

En primer lugar, a Juan Carlos por haberme dado la oportunidad de realizar este proyecto, por todas las ayudas que me ha proporcionado, y todas las dudas que me ha solventado durante su desarrollo.

A mis amigos y compañeros con los que he compartido estos años de universidad, en especial a Antonio y Raed, pues gracias a su compañía y su amistad he logrado minimizar los momentos difíciles y potenciar los buenos y a mi amigo Dani, que ha sido capaz de solventarme la incontable cantidad de dudas con las que he recurrido a él.

A mis amigos de mi instituto y de mi barrio, que siempre me han apoyado y han demostrado estar ahí cuando más los necesitaba.

Y, por último, a mi familia, que son sin ninguna duda, los que más han tenido que lidiar con mi ansiedad y con los peores momentos, y los que a la vez me han proporcionado el mayor empujón de todos para continuar adelante, y los que, finalmente, más orgullosos de mi se han sentido cuando todo salió bien.

INDICE DE CONTENIDOS

1 Introducción

1.1 Motivación

Esta memoria de TFG resume el estudio del modelo óptimo para clasificar el estado de estrés de los usuarios de smartwatch. Trata de mejorar el sistema de identificación actual, el cual está clasificándolo únicamente por la superación de una determinada unidad de pulso cardiaco.

El sistema actual está integrado en una aplicación para dispositivos móviles y tabletas. La aplicación es capaz de detectar el estrés del usuario que lleva el smartwatch gracias a una cifra fija, y en caso de detección dará una serie de instrucciones de relajación con el fin de solventar la crisis.

Las pruebas y el uso de la aplicación han tenido un resultado favorable, sin embargo, siendo posible realizar una optimización y una mejora de este sistema, se ha propuesto realizar una predicción de este estado de estrés. Utilizando el aprendizaje automático, cada detección del estrés tendrá como base los datos de cada persona individualmente, solucionando problemas como la incapacidad de distinguir entre una situación de estrés del ejercicio físico.

Gracias al aprendizaje automático, dispositivos como ordenadores, móviles y relojes son capaces de analizar y categorizar datos mejor y más rápido que cualquier persona, hasta el punto de realizar predicciones futuras de datos con una tasa de acierto considerable que, a simple vista, parecen imposibles de adivinar y obtenerlas.

1.2 Objetivos

El objetivo final de este proyecto es la búsqueda del método de mayor eficiencia de identificar el estrés en base a los datos proporcionados por un smartwatch. Para ello, será necesario encontrar el modelo de clasificador más eficiente para este conjunto de dato.

Para llegar al objetivo final, es necesario realizar una serie de pasos con su propia finalidad individual, de forma que nos acerque un poco más a nuestro objetivo final.

Optimización y adaptación del conjunto de datos, para que sea entrenado por los distintos modelos, el resultado de ello será un conjunto de datos que no contenga datos nulos y sin aquellos datos que pudiesen ser conflictivos.

Selección de modelos candidatos a mostrar sus resultados, el objetivo es realizar una selección coherente, un ejemplo es la de haber descartado la red neuronal, ya que implicaba el uso de demasiados recursos del dispositivo.

Toma de medidas para evitar el sobreentrenamiento, se realizará una división de los datos en particiones de entrenamiento frente a particiones de clasificación, provocamos que el conjunto de datos sea dinámico, evitando que memorice los datos en cada clasificación. Obtendremos un número de particiones de cada tipo, justificado de forma que se realice una clasificación lo más cercana a situaciones reales.

Evaluación de la clasificación, tras entrenar las particiones de entrenamiento y clasificar el resto, se realizará un recuento de los aciertos y fallos de las predicciones con los datos reales. El modelo que mayor porcentaje de acierto tenga será el seleccionado.

Para cada usuario, se han tomado datos del pulsímetro, giroscopio, acelerómetro del detector de pasos del smartwatch en un determinado momento, en caso de que la predicción del modelo coincida con la realidad, se considerará un acierto.

Un objetivo un paso más lejos de este proyecto, es la utilización del clasificador que se ha concluido como el más optimo, esta tarea corresponderá al responsable de programación de la aplicación, quien deberá integrar dicho clasificador al módulo de la aplicación que actualmente evalúa el estado de estrés o relajación mediante el valor del pulso cardiaco.

1.3 Organización de la memoria

La memoria consta de los siguientes capítulos:

- **Introducción:** En el primer capítulo se realiza una primera vista global del proyecto, donde se resume que tipo de contenido es el que vendrá a continuación, la idea que llevó a ello y los objetivos a alcanzar.

- **Estado del arte:** En el segundo capítulo se hablará del estado del arte actual que engloba a la tecnología estudiada durante todo el proyecto, el aprendizaje automático.

- **Diseño:** En el tercer capítulo se definirá el proyecto, se mostrará que alcance tiene y su funcionalidad. Se hablará de los de los entornos de programación, lenguaje y librerías elegidos y cuál y el motivo de su utilización. Por último, se mostrará el ciclo de vida del trabajo.

- **Desarrollo:** En el cuarto capítulo se explicará sobre la implementación del proyecto.

- **Pruebas y resultados:** En el quinto capítulo se expondrán las pruebas utilizadas para visualizar el correcto funcionamiento de los clasificadores, y los resultados que se han obtenido.

- **Conclusiones y trabajo futuro:** En el último capítulo se verán las conclusiones sobre todo lo ya realizado y los pasos a dar tras la finalización del proyecto.

2 Estado del arte

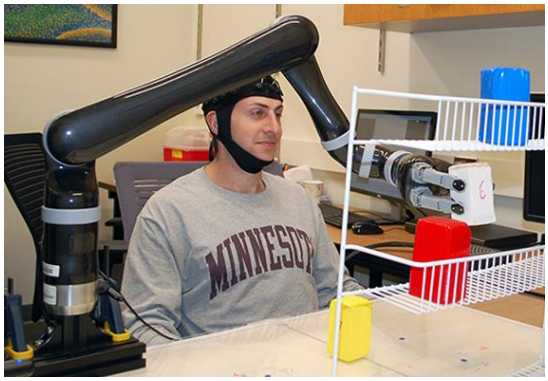
2.1 El aprendizaje automático

El aprendizaje automático es una de las ramas de la inteligencia artificial, la cual se centra esencialmente en el uso de técnicas que permite a los ordenadores y dispositivos computacionales aprender información y utilizarla para resolver problemas alcanzables con la misma.

En esencia, el objetivo del aprendizaje automático es generalizar o inducir reglas desconocidas a partir de datos donde esa regla es aplicada.

Algunos de los ejemplos prácticos más conocidos son las aplicaciones de enseñanza de idiomas, como Duolingo, aquellas que han optado por incluir aprendizaje automático, permiten que el dispositivo que está proponiéndote ejercicios y repeticiones del idioma a aprender se adapten a cada persona identificando sus puntos fuertes y débiles para hacer más eficiente la enseñanza.

Los límites del aprendizaje automático son actualmente indescriptibles, pues actualmente se está aplicando en proyectos del grado de controlar un ordenador o dispositivo electrónico por medio de enseñar al computador a reconocer impulsos neuronales que provoca el cerebro, o controlar piezas del cuerpo robóticas capaces de responder a esos impulsos de forma que permitan afrontar una discapacidad. Kimberley Mok explica en su artículo *How to Control a Robotic Arm with Your Mind, by Using Machine Learning*, cómo con impulsos neuronales entrenados mediante aprendizaje automatico, son capaces de mover un brazo robótico y de manipular objetos.



Para conseguir este tipo de resultados hay dos formas de afrontar el aprendizaje de datos, de forma supervisada o de forma no supervisada. La forma supervisada permite realizar conclusiones que previamente han sido incluidas en estos datos, como es el caso del algoritmo diseñado en este trabajo de fin de grado. La forma no supervisada, permite clasificar distintos conjuntos de datos sin definir como de satisfactorio es el resultado, por ejemplo, podría predecir si una imagen es distinta de otra, pero sería incapaz de definir que es cada imagen.

3 Diseño

Para realizar la búsqueda del clasificador más óptimo se han realizado dos programas independientes, cada uno de ellos tiene sus propios requisitos funcionales y sus propias características, pues el propósito de cada uno de ellos es distinto.

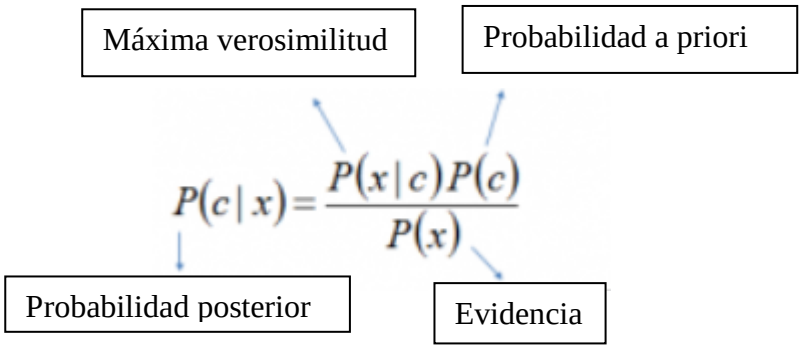
3.1 Modelos de clasificación

- Naive Bayes

Utiliza el teorema de Bayes para realizar un cálculo de probabilidad. Este clasificador considera que cada característica de los datos del conjunto influye en la clasificación de forma independiente. Un ejemplo es que una fruta puede considerarse como manzana si es roja, amarilla o verde, tiene unos 7 centímetros de diámetro, etc.

La clasificación de cada dato se basará en aquella clase cuya probabilidad posterior sea mayor.
Una ventaja de este clasificador es que con un poco cantidad de datos de entrenamiento es capaz de ser muy eficiente.

La fórmula para calcular la probabilidad a posteriori es



- KNN

El método de clasificación KNN es no paramétrico, y estima el valor de la probabilidad posterior en base a la información proporcionada por el resto de los datos, en base a una distancia una vez situados en el espacio.

Para realizar el calculo se establece una *k* que recoge el número de datos que el clasificador usará para estudiar el porcentaje de similitud con las posibles clases del conjunto. El sobreajuste que conlleva utilizar este clasificador viene dado por el número de vecinos dado. Cuanto menor sea el numero de vecinos, mayor es el sobre ajuste, y cuanto mayor sea el número de vecinos, mayor será la similtud con un clasificador basado en la probabilidad a priori de los datos.

- Regresión logística

Los modelos lineales, también pueden ser utilizados para clasificaciones; para ello, que primero ajustamos el modelo lineal a la probabilidad de que una cierta clase o categoría ocurra y, a luego, utilizamos una función para crear un umbral en el cual especificamos el resultado de una de estas clases o categorías. La función que utiliza este modelo no es ni más ni menos que la función logística.

$$f(x) = \frac{1}{1 + e^{-1}}$$

- Árbol de decisión

Los Arboles de Decision son diagramas con construcciones lógicas, parecidos a sistemas de predicción que se basan en reglas, representan y clasifican en categorías una serie de características que ocurren sucesivamente, con el objetivo de solventar problemas.
Los Arboles de Decision están compuestos por nodos interiores, ramas que emanan de los nodos interiores, y nodos terminales. Cada nodo interior en el árbol contiene un atributo, y cada una de las ramas del nodo, representan los posibles valores que puede coger ese atributo.
Siguiendo el camino representado por las ramas, el final de estas es un nodo terminal que crea una segmentación de los datos.

- Random Forest

En lugar de utilizar un solo arbol para realizar la clasificación, se utiliza un bosque, es decir un conjunto de árboles.
Y este es el sistema que crea Random Forest, trabaja con distintos arboles de decisión de una profundidad muy baja, y hace un balance entre las clases que cada uno de los arboles ha escogido. Esta idea es muy poderosa en Machine Learning. Si tenemos un clasificador que tiene un porcentaje de acierto notable, podríamos entrenar una gran cantidad de clasificadores, que predigan nuestros datos y que el resultado fuese la clase predominante de todas las predicciones.

3.2 El problema del sobreentrenamiento y cómo evitarlo

El sobreentrenamiento es la inclinación que tienen los algoritmos de aprendizaje automático a acostumbrarse a las reglas específicas que contiene un conjunto de datos en concreto. El algoritmo tiende a “memorizar” las respuestas válidas que el conjunto le proporciona, de forma que obtiene un alto rendimiento clasificando ese conjunto de datos, pero se vuelve incapaz de clasificar datos nuevos, los cuales mantienen los atributos de los datos entrenados, pero son ajenos a su conjunto.

Para evitar este problema es necesario que el algoritmo clasifique datos que no aparecen en su modelo de datos. Solo de esta forma podemos asegurar que el algoritmo de clasificación

escogido realmente ha encontrado el método de predicción de los nuevos datos, y no ha caído en el sobreentrenamiento.
Para ello, en esta búsqueda del clasificador más eficiente para clasificar modelos de estrés y relajación, vamos a dividir los datos en particiones y haremos uso de la **validación cruzada**.

Utilizaremos un porcentaje de nuestro conjunto de datos (varias particiones del mismo tamaño) para entrenar nuestro modelo de clasificación, y el porcentaje restante (el resto de las particiones) serán los datos a clasificar. Pero gracias a la validación cruzada, las particiones utilizadas para entrenamiento y las particiones utilizadas para clasificación estarán en rotación, es decir, una partición no va a estar sometida a entrenamiento en todas las pruebas, sino que, mediante cambios sistemáticos, existirán pruebas en las que una determinada partición servirá como entrenamiento, y existirán pruebas donde esta será clasificada.

Lo veremos en más detalle en los siguientes puntos de este mismo capítulo.

3.3 Scikit-learn

Scikit-learn es la librería que permite que todo el proyecto funcione, otorga herramientas para el análisis y minería de datos y con los conocimientos necesarios, ayuda a que realizar clasificaciones por medio de aprendizaje automático sea más sencillo.

Es una librería que incluye todos los modelos utilizados para realizar entrenamiento y clasificación de este proyecto, así como, herramientas de validación como kfold (Validación cruzada). Además, tiene total integración con otras librerías de tratamiento de datos como numpy, pandas, SciPy, y matplotlib.
Estás librerías que scikit-learn es capaz de manejar junto a sus propias herramientas han sido las que hacen posible todo el previo tratamiento de los datos con respecto a la clasificación, ya que normalmente, los datos provienen de ficheros externos que deben ser adaptados para su correcto uso en los algoritmos de aprendizaje automático.

3.4 Análisis y procesado de datos

El módulo de Análisis y procesado de datos tiene como objetivo procesar el conjunto de datos dado y transformar dicho conjunto de forma que sea asimilable por el segundo programa. Para ello, clasifica cada dato como numérico o categórico, ya que a cada tipo de dato es necesario tratarlo de distinta forma, y trata los valores nulos. Se plantearon tres alternativas a la hora de actuar frente a los datos nulos:

- No hacer nada, se descubrió que a la larga traería demasiados problemas, ya que los algoritmos de clasificación tienen problemas al procesar datos nulos, especialmente en los datos numéricos, que no identifica el dato nulo como su mismo tipo de dato (entero, float, etc)
- Eliminarlos, es una gran alternativa, pero en conjuntos de datos donde en la mayoría de los datos se contiene algún atributo que se ha recogido como nulo, nos destruiría en gran medida el conjunto de datos.
- Darle un nuevo valor, que es la alternativa que se ha seguido. El motivo es que cada valor nulo puede ser juzgado observando los valores del mismo atributo del resto

de datos, y se puede provocar que este dato originalmente nulo, tenga una relevancia insignificante para el calificador. A continuación, se explicará de que forma se han modificado dichos datos

Se identifica cada tipo de dato de cada atributo, y se dividen en atributos numéricos y atributos nominales.

Para los datos nulos de atributos numéricos se aplica la media de los valores de los demás datos de ese mismo atributo, esto sirve porque a la hora de que el clasificador aprenda y asimile ese valor, estar en la media es considerado como no decisivo para dar el resultado de la clasificación del dato.

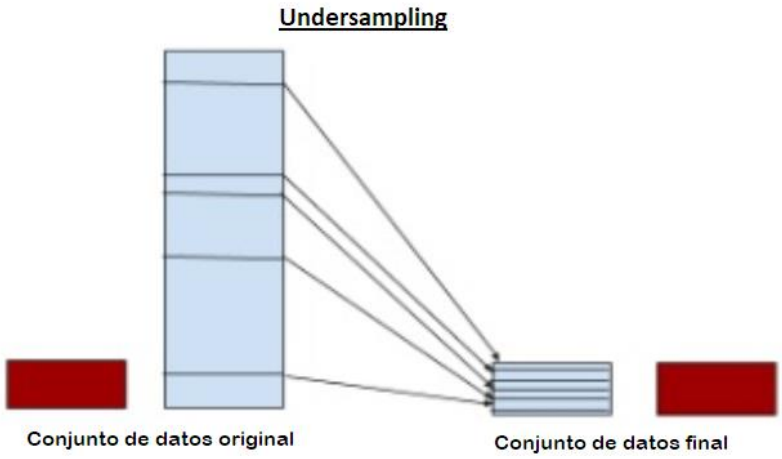
La decisión tomada en cuanto a los datos nulos de atributos nominales es la de aplicar la moda, la cual podría ser más cuestionable, y ha sido necesario justificarla y ha sido gracias al estilo del conjunto de datos. Se ha decidido tomar la moda del atributo en lugar del dato nulo considerando que ese atributo es relevante para la clasificación, pero que no garantiza que sea crucial para ello.
Un ejemplo de esto podría ser un atributo como el Género, que puede tomar valor 'masculino' o 'femenino'. En caso de encontrar un dato nulo, y dada la imposibilidad de dejar el cambo en blanco (asumimos que únicamente puede tomar uno de los dos valores), se establece la posibilidad mayor, pues será la que tiene menos posibilidad de estar equivocada, y esta es, la moda de los valores ese atributo. En resumen, en caso de la falta del dato de género, se establece, por ejemplo, como femenino, por haber sido el resultado de la moda de este atributo.

3.5 Clasificación de datos

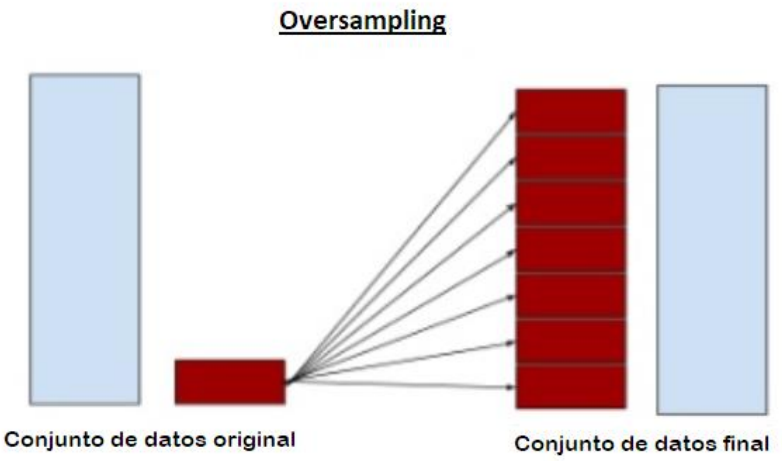
El segundo módulo es realiza la clasificación de los datos es el programa que da el resultado del estudio estableciendo un porcentaje de acierto a cada partición clasificada.

Para ello, recoge los datos del dataset procesado, resultado del programa anterior, y utilizando las funciones de la librería de pandas. Tras recoger el dataset será necesario dividir los datos entre datos de entrenamiento y clase a predecir.
Una de las características del dataset entregado es la existencia de una clase muy mayoritaria a la otra, esto es un problema grave, ya que puede dar el resultado de una clasificación muy válido únicamente colocando todos los resultados al valor de la clase mayoritaria. Para solventar este problema, se han realizado dos estrategias y han sido aplicadas en ejecuciones distintas con el objetivo de comparar los resultados y seleccionar el que mejor funcionase, estas estrategias son el uso del **undersampling** y del **oversampling**.

El undersampling solventa el problema de la diferencia de clases de una forma más sencilla, pero provoca que el resultado sea más impreciso. Esto ocurre porque el undersampling selecciona un número de dados que contienen la clase mayoritaria, y los elimina hasta que ambas clases contienen el mismo número de datos.



Por el contrario, el oversampling aumenta los datos de la clase minoritaria hasta alcanzar a la mayoritaria, para ello, utiliza un algoritmo de mutación sobre los datos de la clase minoritaria, creando nuevos posibles datos de esta clase con atributos similares a los que pertenecen a esta clase.



A continuación, se realizará un array de modelos a estudiar. En este caso se han añadido al array los modelos de Naive Bayes, KNN con 3 vecinos, 5 vecinos y 7 vecinos, Regresión Logística, Árbol de Decisión y Random Forest.

Por cada modelo del array se realizará un entrenamiento de los datos, y una clasificación mediante validación cruzada. La validación cruzada escogida divide los datos en 5 particiones con los datos previamente "barajados" y utiliza en cada una de las 5 iteraciones, cuatro particiones de entrenamiento, y una de test.
Por ejemplo, en la primera iteración, las particiones B, C, D, E y F, se utilizarán como entrenamiento, y la, la partición A sería la utilizada para clasificar y ser evaluada; en la siguiente iteración, serán las particiones A, C, D, E, y F las utilizadas como entrenamiento, y la partición B será la utilizada para clasificar y ser evaluada. Se procederá de la misma forma con las tres iteraciones siguientes hasta que el dataset al completo ha sido clasificado y evaluado.

Por último, por cada partición se decidirán si los datos van a ser entrenados (Train) o clasificados (Test), es importante conocer que datos van a ser clasificados para guardar las clases que tienen en un primer lugar, de esta forma podremos hacer la comparación entre el resultado y como es en realidad, y de ahí sacar el porcentaje de acierto.

Utilizamos funciones de scikit-learn para entrenar con el modelo de la iteración actual y para realizar la clasificación, finalmente, creamos una matriz de confusión de los resultados de cada partición.

Una matriz de confusión es una representación de los resultados, colocados en forma de matriz; en cada columna se ve representado el número de predicciones de cada valor de la clase, y en cada fila se representa a las el valor real, de esta forma es muy fácil identificar los falsos negativos y los falsos positivos.

3.6 Análisis de requisitos

- **Requisitos Funcionales**

  RF1: Formato y contenido del fichero de entrada.

  El formato del fichero a procesar debe ser .csv, contener una cabecera de una línea que contiene los nombres de cada atributo, los atributos deben estar separados entre sí por el separador ‘;’ y cada dato se representa en una línea distinta. La clase debe ser el último atributo de cada dato.

  RF2: El fichero resultante del módulo de procesado.

  El nuevo fichero no contendrá datos nulos que pudiesen provocar errores en la futura clasificación, y el nombre del fichero mantendrá el nombre original junto a la palabra ‘procesado’, para identificarlo fácilmente.

  RF3: Fichero de entrada al módulo de clasificación.

  El formato del fichero a procesar debe haber pasado por el procesador de datos, de esta forma, el buen funcionamiento del programa está asegurado. En caso de que el fichero no se encuentre en la ubicación determinada se indicará que no ha sido encontrado y parará la ejecución.

RF4: Uso de los modelos de clasificación
Los modelos de clasificación deben entrenar las particiones generadas por la validación cruzada y generar unos resultados por cada partición

RF5: Salida del programa

El módulo de clasificación debe mostrar los resultados de cada clasificación de forma ordenada, indicando que partición es la clasificada y con su respectiva matriz de confusión.

- **Requisitos no funcionales**

RNF1: Estilo de programación y metodología coherente.

Como el proyecto debe servir de guía para el encargado de trasladar el módulo de clasificación a la aplicación en Android, es imprescindible que tenga la capacidad de leer el código, y de entenderlo.

RNF2: Flexibilidad del código

A pesar de que el programa haya sido creado con el propósito de solventar el objetivo de este proyecto concreto, el código debe poseer la flexibilidad de clasificar otros modelos de datos distintos, de forma que al necesitar un proyecto similar para nuevos datos, no sea necesario empezar de cero.

RF3: Rendimiento de los modelos de clasificación.

El principal motivo de que no se realicen pruebas de por ejemplo, redes neuronales, es que, al realizar en un trabajo futuro, la integración con smartwatches y aplicaciones móviles corre el riesgo de no ser soportado. Es por ello que los clasificadores elegidos a 'competir' entre ellos, tienen un alto rendimiento para los recursos que necesitan.

RNF4: Robustez

El programa debe contener un sistema mínimo de control de errores y de seguridad.

3.7 Ciclo de vida

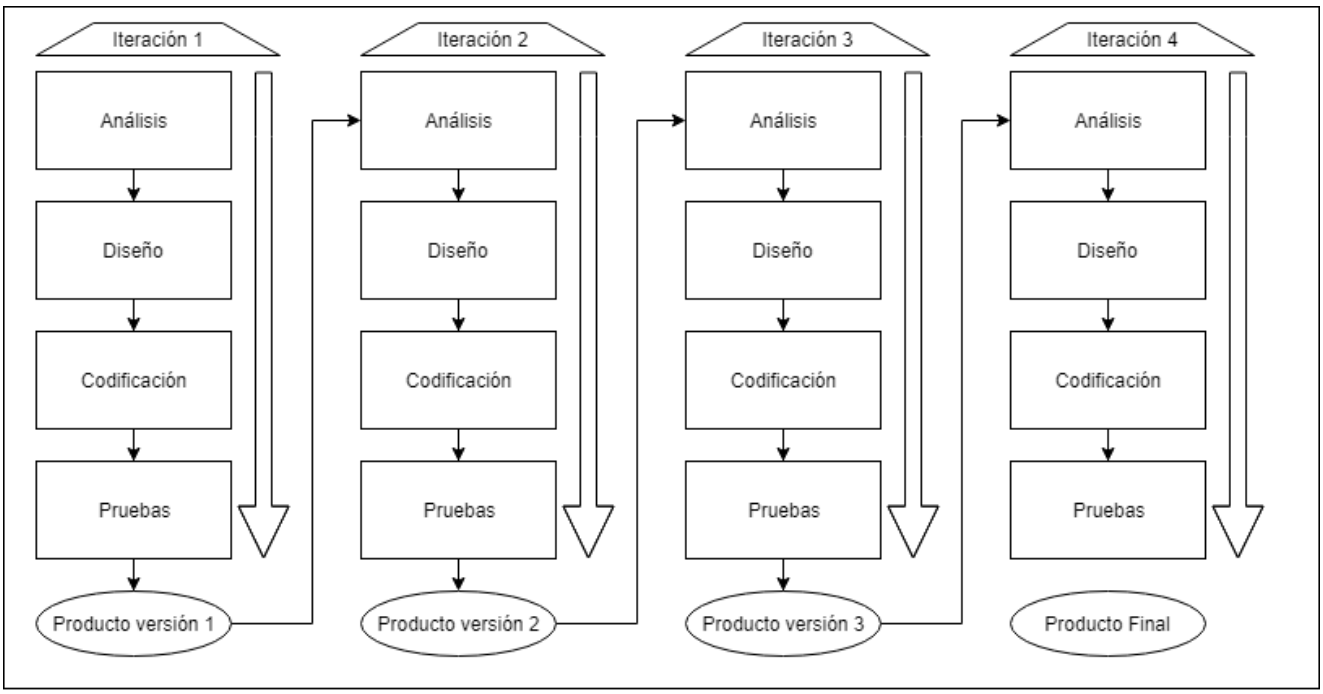
Este proyecto sigue un ciclo de vida incremental, en cada iteración, el proyecto ha sufrido mejoras en todas sus fases hasta dar como resultado el programa final.

Iteración 1: Diseño e implementación del programa de forma básica. Procesado de datos eliminado del conjunto directamente todos los datos que resultaban conflictivos. Clasificación de los datos sin validación cruzada, y únicamente con un modelo de clasificación. Resultados sin matriz de confusión.

Iteración 2: Mejora en la identificación de datos conflictivos, identificación de datos categóricos y cuantitativos. Aplicación de la validación cruzada al clasificador implementado. Muestra de resultados por particiones.

Iteración 3: Procesado completo de datos, concluyendo cada método utilizado según el tipo de dato. Declaración de nuevos clasificadores y estudio del consumo de recursos y tiempo de clasificación, estudio del problema de la clase mayoritaria. Muestra de los resultados de cada clasificación.

Iteración 4: Elección de los clasificadores finales e implementación de los métodos de undersampling y oversampling. Muestra de la matriz de confusión junto a los resultados.



4 Desarrollo

Al igual que en el apartado de diseño, el desarrollo ha sido dividido en dos secciones, cada una corresponde a los dos programas que hacen que el estudio del clasificador más optimo se realice.

4.1 Procesador de datos

4.1.1 Librerías utilizadas

- Pandas: Nos permite realizar operaciones con conjuntos de datos, manejo de estos por filas y columnas, generación y extracción de datos de ficheros csv.
- Numpy: Nos permite tener un manejo de los arrays de datos más completo que pandas, permitiendo aplicar funciones de algebrar lineal, consultar los tipos de datos, sustitución de nulos a aquellos que cumplan determinada condición, etc.

4.1.2 Desarrollo del programa

Es importante recordar que en la etapa de análisis y diseño se ha realizado un pequeño estudio de como está construido el conjunto de datos a clasificar. Una vez lo conocemos superficialmente, vamos a procesarlo.

En primer lugar, lo más importante es importar el conjunto de datos a nuestra variable de tipo DataFrame de la librería de pandas, esto nos va a permitir acceder a cada dato de forma sencilla.

```
import numpy as np
import pandas as pd

print("Start")
# Importar el conjunto de datos
dataset = pd.read_csv("datos/all_users_nulos.csv", header=0, converters={object: str}, sep=';')
```

Para tener una visión de los datos durante la ejecución vamos a mostrar los 10 primeros datos de algunos de los atributos y para controlar que los datos no varían durante toda la ejecución comprobamos cuantos datos tiene el conjunto.

[In]:

```
# Vista de las primeras 10 filas de algunos de los atributos
print(dataset.ix[:9, 15::15])

rows, columns = dataset.shape
# Cantidad de datos
print("Numero de datos:", rows)
# Cantidad de atributos
print("Numero de atributos:", columns)
```

Esto nos proporciona la siguiente salida:

[Out]:

	STDV_MPU6515_Accelerometer_Y	MAD_MPU6515_Accelerometer_Z	MEDIAN_MPU6515_Gyroscope_X
0	3.979095	4.974163	0.215240
1	5.053516	3.162408	0.231689
2	2.037963	0.334921	0.009094
3	3.693800	0.627520	0.047318
4	0.757911	5.298721	0.043228
5	0.951681	0.691377	NaN
6	0.663340	1.298328	0.002243
7	0.462432	1.047140	NaN
8	1.356710	1.370868	-0.004272
9	0.990884	0.993791	0.009003

	COEF_2^1_MPU6515_Gyroscope_Y	COEF_2^3_MPU6515_Gyroscope_Z	moc_Heart_Rate_Monitor
0	5.627009e-05	1.936670e-13	1.0
1	-9.620871e-07	-1.257448e-13	3.0
2	2.122555e-04	2.644935e-14	NaN
3	3.632006e-04	-2.275784e-11	NaN
4	4.139935e-05	-4.017058e-11	NaN
5	NaN	NaN	NaN
6	-3.883693e-05	-2.871540e-11	NaN
7	NaN	NaN	NaN
8	-9.095306e-05	-1.398845e-11	NaN
9	6.080911e-04	1.154791e-10	NaN

```
Numero de datos: 5575
Numero de atributos: 105
```

Esta carga de datos utilizando la librería de pandas nos permitirá realizar este tratamiento a los datos con facilidad.
El siguiente paso que debemos dar es el de controlar el número de datos nulos, ya que son realmente perjudiciales para los clasificadores:

[In]:

```
# Identificar si existen valores Nulos
print("Valores nulos:",dataset.isnull().any().any())
```

[Out]:

```
Valores nulos: True
```

Vamos a realizar un tratamiento distinto según el tipo de atributo, agruparemos las columnas del conjunto de datos identificando si son numéricos o nominales:

[In]:

```
# Agrupando columnas por tipo de datos
tipos = dataset.columns.to_series().groupby(dataset.dtypes).groups
# Creando listado de atributos numericos
cnum = tipos[np.dtype('float64')]
# Creando listado de atributos nominales (El resto)
totalAtributos = dataset.columns
ctext = list(set(totalAtributos) - set(cnum))

print("Atributos numericos:", len(cnum))
print("Atributos nominales:", len(ctext))
```

[Out]:

```
Atributos numericos: 104
Atributos nominales: 1
```

Se puede destacar que, en nuestro conjunto de datos, únicamente existe un atributo de tipo nominal, la clase, el atributo que será el clasificado tras procesar todo el conjunto de datos.

Los datos numéricos nulos provocarían un error en el conjunto de datos porque el dato nulo no pertenece al mismo tipo de dato que el numérico, el cual podría ser tipo float, entero, double, etc.

Los datos nominales nulos, podrían ser identificados como una nueva categoría, dando por hecho que todo dato nulo pertenece a la misma categoría de dato. Por ejemplo, si en un determinado atributo nominal existen las categorías, 'perro' y 'gato', el clasificador entendería que existen las categorías 'perro', 'gato' y 'ni perro ni gato', siendo esta última, el dato Nulo.

Como se ha explicado en el apartado de diseño, se ha optado por modificar el valor de los datos nulos de forma que no influyan de forma relevante en la clasificación, y para ello vamos a sustituir los valores nulos de los datos numérico por la media de los valores del atributo, y lo valores nulos de los datos nominales por la moda de los valores del atributo.

```
# Completar los datos NULOS
# Usando la media con los numéricos
for c in cnum:
    meanf = dataset[c].mean()
    dataset[c] = dataset[c].fillna(meanf)

# Usando la moda con los nominales
for c in ctext:
    mode = dataset[c].mode()[0]
    dataset[c] = dataset[c].fillna(mode)
```

Tras la eliminación de valores nulos dentro de atributos, eliminamos completamente aquellos que no contienen ningún valor no nulo en ningún dato para el mismo atributo

[In]

```
# Borrando atributos completamente nulos
dataset = dataset.dropna(axis=1,how='all')

# Comprobación de que ya no existan valores NULOS
print("Valores Nulos:",dataset.isnull().any().any())

print(dataset.ix[:9, 15::15])
```

[Out]

	STDV_MPU6515_Accelerometer_Y	MAD_MPU6515_Accelerometer_Z	MEDIAN_MPU6515_Gyroscope_X
0	3.979095	4.974163	0.215240
1	5.053516	3.162408	0.231689
2	2.037963	0.334921	0.009094
3	3.693800	0.627520	0.047318
4	0.757911	5.298721	0.043228
5	0.951681	0.691377	9.159250
6	0.663340	1.298328	0.002243
7	0.462432	1.047140	9.159250
8	1.356710	1.370868	-0.004272
9	0.990884	0.993791	0.009003

	COEF_2^1_MPU6515_Gyroscope_Y	COEF_2^3_MPU6515_Gyroscope_Z	moc_Heart_Rate_Monitor
0	5.627009e-05	1.936670e-13	1.000000
1	-9.620871e-07	-1.257448e-13	3.000000
2	2.122555e-04	2.644935e-14	8.462673
3	3.632006e-04	-2.275784e-11	8.462673
4	4.139935e-05	-4.017058e-11	8.462673
5	3.618479e-02	2.118976e-07	8.462673
6	-3.883693e-05	-2.871540e-11	8.462673
7	3.618479e-02	2.118976e-07	8.462673
8	-9.095306e-05	-1.398845e-11	8.462673
9	6.080911e-04	1.154791e-10	8.462673

```
Valores Nulos: False
```

Y finalmente, guardamos el nuevo conjunto de datos

```python
# Guardamos el nuevo conjunto de datos
dataset.to_csv("datos/all_users_procesado.csv", index=False)
```

4.2 Clasificador

4.2.1 Librerías utilizadas

- Pandas: Nos permite realizar operaciones con conjuntos de datos, manejo de estos por filas y columnas, generación y extracción de datos de ficheros csv.
- Numpy: Nos permite tener un manejo de los arrays de datos más completo que pandas, permitiendo aplicar funciones de algebrar lineal, consultar los tipos de datos, sustitución de nulos a aquellos que cumplan determinada condición, etc.
- Sklearn: Utilizada por que nos proporciona cada uno de los algoritmos para entrenar y clasificar con los modelos acordados en el diseño, además incluye funciones para realizar la validación cruzada y pintar la matriz de confusión.
- Imblearn: Utilizada por que nos proporciona los algoritmos de tratamiento de datos que aplican undersampling y oversampling.

4.2.2 Desarrollo del programa

En la etapa de análisis y diseño, y en la del desarrollo del programa anterior se ha estudiado de que forma ha resultado el conjunto de datos a clasificar. Una vez lo conocemos hemos procesado, vamos a realizar la clasificación.

En primer lugar, lo más importante es importar el conjunto de datos a nuestra variable de tipo DataFrame de la librería de pandas, esto nos va a permitir acceder a cada dato de forma sencilla.

```
import numpy as np
import pandas as pd

from sklearn.cross_validation import *

from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix

from imblearn.under_sampling import NearMiss
from imblearn.over_sampling import SMOTE

print("Start")
# Importar el conjunto de datos
df = pd.read_csv('datos/all_users_procesado.csv', header=0)
```

Como ya conocemos el conjunto de datos por la implementación del programa de procesado, únicamente vamos a seguir la pista de los datos a lo largo de su ejecución por el número de datos que contiene, esto es porque como veremos más adelante, hay que realizar una nueva pequeña modificación antes de clasificar.

[In]

```
# Dividimos los datos entre atributos y clase
dfX_o = df.ix[:,:-1]
dfy_o = df.ix[:,-1]

# Comprobamos el número de los datos originales
print ("Datos originales")
print (len(dfX_o))
```

[Out]

```
Datos originales
5575
```

A continuación, declararemos nuestros modelos, y serán guardados en un array de modelos, en este caso, utilizamos un diccionario por que nos permitirá identificar cada modelo declarado con el nombre personal dado. Esto nos facilitará mucho identificar la salida del programa.

```
# Declaramos los modelos
models = {}
models["Naive Bayes"] = GaussianNB()
models["2-Near Neighbours"] = KNeighborsClassifier(n_neighbors=2)
models["5-Near Neighbours"] = KNeighborsClassifier(n_neighbors=5)
models["7-Near Neighbours"] = KNeighborsClassifier(n_neighbors=7)
models["Logistic Regresion"] = LogisticRegression(random_state=1)
models["Arbol de Decision"] = DecisionTreeClassifier()
models["Random Forest"] = RandomForestClassifier()
```

Como ya hemos explicado en el apartado de diseño, nuestro conjunto de datos tiene una peculiaridad, se trata de la diferencia del numero de datos que contiene cada clase. Debido al pequeño número de datos que contienen la clase de ‘estrés’, el clasificador entiende que sería muy eficiente predecir todos los resultados del conjunto a ‘no estrés’, pues el porcentaje de fallo de la salida sería tan pequeño como el porcentaje de datos de clase minoritaria frente al de mayoritaria. Pero esto es un error grave, pues el objetivo del proyecto es encontrar el clasificador que mejor encuentra el ‘estrés’ con el menor número de falsos positivos posible.

Es por lo que hemos realizado dos ejecuciones independientes del programa, cada una de ellas aplica una solución distinta al problema anterior.

4.2.2.1 Primera solución, undersalmpling

Consiste en la eliminación de datos sobrantes de la clase mayoritaria, para ello probado dos algoritmos de la librería imblearn llamados, NearMiss y RandomUnderSampler.

Tras comprobar los resultados de ambos, nos decantamos por NearMiss. Ya que éste selecciona los datos eliminados con criterio para no retirar información al clasificador, mientras que el RandomUnderSampler elimina los datos al azar.

```
# Aplicamos UnderSampling sobre los datos
dfX_u, dfy_u = NearMiss(random_state=32).fit_sample(dfX_o, dfy_o)
```

4.2.2.2 Segunda solución, oversampling

Consiste en la adición de datos de la clase minoritaria por medio de mutaciones hasta igualar la mayoritaria, para ello probado un algoritmo de la librería imblearn llamado, Smote.

Smote identifica la clase minoritaria, y crea nuevos datos sintéticos, para ello, duplica los datos antiguos y aplica un algoritmo de mutación sobre estas copias, rotando su posición en el espacio (usando sus atributos para distribuirlos) y alterando los datos de forma que encajen de forma lógica con los actuales.

```
# Aplicamos OverSampling sobre los datos
dfX_u, dfy_u = SMOTE(kind='borderline1').fit_sample(dfX_o, dfy_o)
```

Tras modificar el conjunto de datos por última vez, podemos comprobar el cambio en el número de datos.

[In]

```
# Comprobamos el estado de los datos modificados
print ("Datos finales")
print (len(dfX_u))

# Reconstruimos el conjunto de datos final
df = pd.DataFrame(dfX_u)
df['class'] = dfy_u
dfX = df.ix[:,:-1]
dfy = df.ix[:,-1]
```

[Out] (UnderSampling)

```
Datos finales
100
```

[Out] (OverSampling)

```
Datos finales
11050
```

Y a partir de aquí, podemos proceder a realizar todo el proceso de clasificación. Este consta de cuatro etapas: La aplicación de la validación cruzada, el entrenamiento, la predicción y la comprobación.

Como nuestro programa desarrolla distintos modelos en una única ejecución, realizamos iteraciones por el array de modelos de clasificación, y a cada uno de ellos se realiza la **aplicación de la validación cruzada** a los datos.

```
# Comienza la clasificación de cada modelo
for model in models:
    print("\n" + model)
    print("------------------------------")
    i = 0

    # Aplicamos la validación cruzada (5 folds)
    kf = KFold(df.ix[:,-1].size, n_folds=5, shuffle=True)
```

Para organizar los datos, la validación cruzada nos devuelve, por cada partición, los índices que ha establecido para entrenar y para predecir. Gracias a estos índices, vamos a separar los datos en tres bloques: bloque de entrenamiento, bloque de predicción y bloque original.

El bloque original contiene las clases originales de los datos que recoge el bloque de predicción.

```
    for p in kf:
        # Dividimos los indices entre entrenados y clasificados
        indicestrain, indicestest = p

        # Dividimos los datos y guardamos los originales del test
        train = df.iloc[indicestrain,:]
        test = dfX.iloc[indicestest,:]
        origintest = df.iloc[indicestest,:]
```

A continuación, se realiza **el entrenamiento** de los datos. Cada uno de los modelos de clasificación que nos proporciona la librería Scikitlearn contiene un método denominado 'fit'. Utilizamos este método desde el modelo de la iteración actual.

```
        # Entrenamos el modelo con los datos de train
        print("\nParticion " + str(i) + ":")
        print("- Start Fit")
        models[model].fit(train.ix[:,:-1],train.ix[:,-1])
```

Tras entrenar los datos realizamos **la predicción**. Guardamos el resultado en una variable, la cual se utilizará para calcular el porcentaje de error que ha resultado de predecir esta partición con el modelo de la iteración actual.

```
        # Realizamos la predicción
        print("- Start Predict")
        predict = models[model].predict(test)
```

Finalmente realizamos **la comprobación** de errores y el cálculo del porcentaje de error de la clasificación, mostramos la matriz de confusión de los resultados, de forma que sabremos si cada error producido es un falso positivo o un falso negativo.

```
        # Comparamos las predicciones con los originales
        print("- Resultados con todos los atributos:")
        error = findErrors(origintest.ix,predict)
        print (error)

        # Mostramos la matriz de confusión
        print("\nMatriz de confusion " + str(i) + ":")
        print(confusion_matrix(list(origintest.ix[:,-1]), predict))
```

Para hacer el calculo de error hemos utilizado la función findErros.

```
# Obtiene el numero de aciertos y errores para calcular la tasa de fallo
def findErrors(datos,pred):
    # Comparamoss la prediccion (pred) con las clases originales y calculamos el error
    acierto, fallo = 0, 0
    for i, j in zip(datos[:,-1], pred):
        if i == j:
            acierto += 1
        else:
            fallo += 1
    return 'Acierto' ,acierto, 'Fallo',fallo
```

Todo este proceso nos ha proporcionado una salida que muestra los resultados de las particiones de cada modelo de clasificación.

[Out]

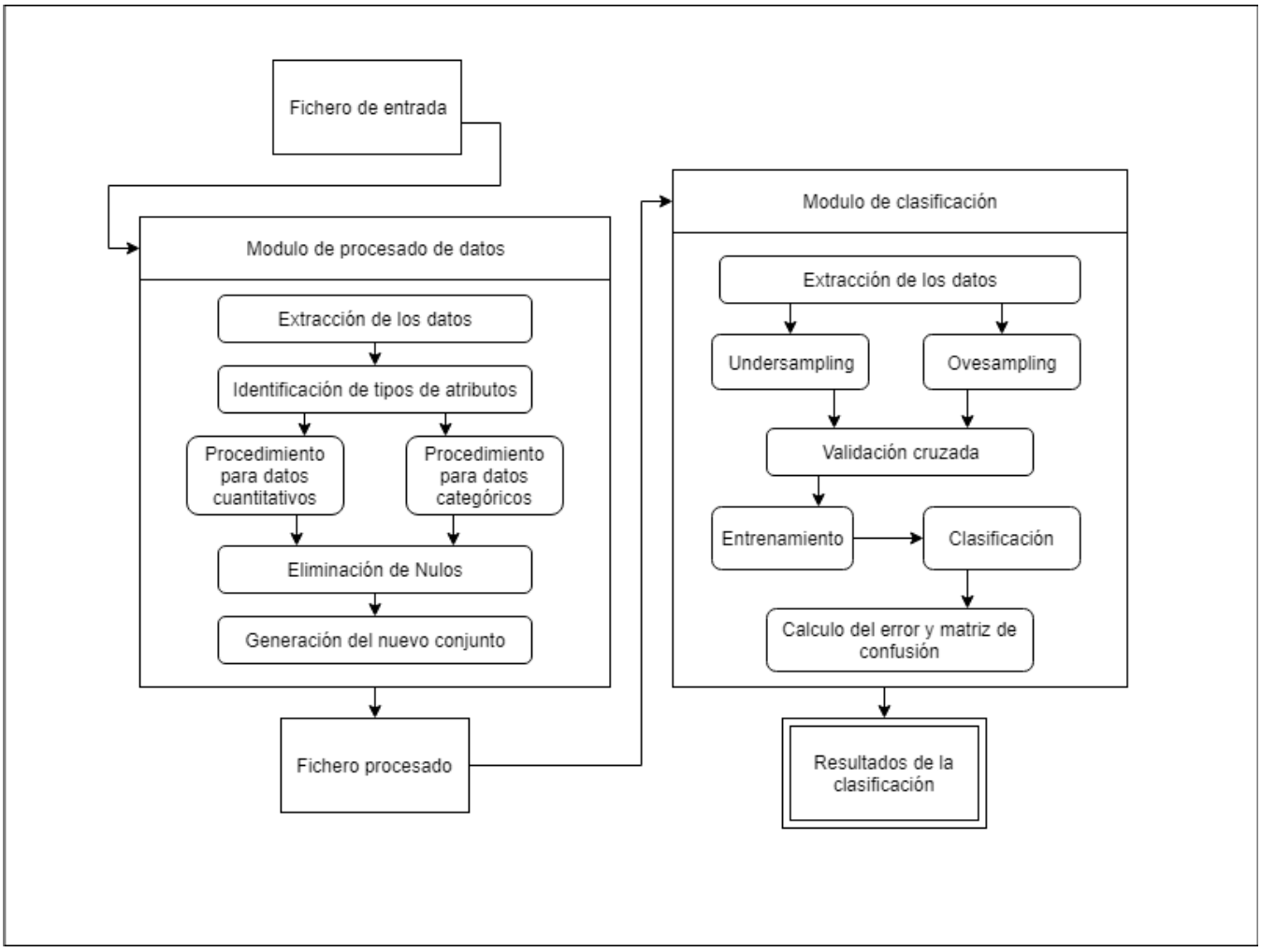
```
Logistic Regresion
----------------------------

Particion 0:
- Start Fit
- Start Predict
- Resultados con todos los atributos:
('Acierto', 1856, 'Fallo', 354)

Matriz de confusion 0:
[[870 240]
 [114 986]]
----------------------------
```

Como se puede comprobar en el ejemplo adjunto, en un principio se muestra el modelo de clasificación dado y a continuación, la partición clasificada, la cual contiene el número de aciertos y errores junto a la matriz de confusión correspondiente.

4.1 Diagrama global del proyecto



En el diagrama se muestran las funciones y el tratamiento que sufren los datos que contiene el fichero de entrada hasta que se obtiene los resultados de la clasificación de dichos datos.

5 Pruebas y resultados

En este apartado, vamos a analizar la salida del programa que nos muestra el resultado de nuestro estudio. Tras recoger los resultados, realizaremos una comparación entre las posibles opciones dadas y concluiremos cual es el mejor clasificador para identificar emociones en los niños con TEA.

5.1 Análisis de los resultados

Como ya explicamos anteriormente, por cada clasificador hemos realizado dos ejecuciones, una de ellas aplicando undersampling y la otra aplicando oversampling. El análisis será realizado teniendo en cuenta ambas ejecuciones, y para ello vamos a plasmar ambas por separado.

También incluiremos la matriz de confusión de cada resultado, pues no solo es importante conocer cuántos fallos y aciertos ha realizado cada clasificador, también lo es conocer que fallos y que aciertos ha cometido para cada posible valor de la clase. En la matriz de confusión podemos observar los datos originales, representados mediante las filas, y los datos clasificados, representados mediante las columnas, lo cual queda final mente así.

Predicción / Original	Estrés original	Relajación original
Estrés predicho	Verdadero positivo	Falso positivo
Relajación predicho	Falso negativo	Verdadero negativo

5.2 Ejecución con undersampling

En esta ejecución, el total de datos predichos es de 100. Al utilizar 5 particiones, cada partición clasifica 20 datos, que han sido clasificados mediante el entrenamiento de los otros 80.

5.2.1 Naive Bayes

Tabla de resultados:

Partición	Acierto	Fallo	Acierto (%)	Fallo (%)
0	14	6	70	30
1	19	1	95	5
2	18	2	90	10
3	17	3	85	15
4	19	1	95	5
Total	*87*	*13*		
Medio	*17,4*	*12.6*	*87*	*13*

Matriz de confusión:

P/O	E	R
E	45	3
R	10	42

5.2.2 KNN

2 VECINOS

Tabla de resultados:

Partición	Acierto	Fallo	Acierto (%)	Fallo (%)
0	17	3	85	15
1	16	4	80	20
2	18	2	90	10
3	18	2	90	10
4	17	3	85	15
Total	*86*	*14*		
Medio	*17,2*	*2,8*	*86*	*14*

Matriz de confusión:

P/O	E	R
E	58	3
R	11	28

5 VECINOS

Tabla de resultados:

Partición	Acierto	Fallo	Acierto (%)	Fallo (%)
0	15	5	75	25
1	19	1	95	5
2	17	3	85	15
3	18	2	90	10
4	18	2	90	10
Total	87	13		
Medio	17,4	2,6	87	13

Matriz de confusión:

P/O	E	R
E	46	2
R	11	41

7 VECINOS

Tabla de resultados:

Partición	Acierto	Fallo	Acierto (%)	Fallo (%)
0	14	6	70	30
1	17	3	85	15
2	18	2	90	10
3	19	1	95	5
4	17	3	85	15
Total	85	15		
Medio	17	3	85	15

Matriz de confusión:

P/O	E	R
E	47	5
R	10	38

5.2.3 Regresión Logística

Tabla de resultados:

Partición	Acierto	Fallo	Acierto (%)	Fallo (%)
0	18	2	90	10
1	15	5	75	25
2	18	2	90	10
3	19	1	95	5
4	15	5	75	25
Total	85	15		
Medio	17	3	85	15

Matriz de confusión:

P/O	E	R
E	50	4
R	11	35

5.2.4 Árbol de decisión

Tabla de resultados:

Partición	Acierto	Fallo	Acierto (%)	Fallo (%)
0	17	3	85	15
1	16	4	80	20
2	17	3	85	15
3	17	3	85	15
4	17	3	85	15
Total	84	16		
Medio	16,8	3,2	84	16

Matriz de confusión:

P/O	E	R
E	39	11
R	5	45

5.2.5 Random Forest

Tabla de resultados:

Partición	Acierto	Fallo	Acierto (%)	Fallo (%)
0	17	3	85	15
1	19	1	95	5
2	18	2	90	10
3	19	1	95	5
4	18	2	90	10
Total	91	9		
Medio	18,2	1,8	91	9

Matriz de confusión:

P/O	E	R
E	48	2
R	7	43

Tras examinar los resultados de las clasificaciones, podemos observar que, en este caso, el clasificador Random Forest, ha resultado ser el más preciso. Incluso teniendo un porcentaje de acierto mayor a un 5% con respecto el Árbol de decisión, que como ya explicamos en el apartado de diseñó, es un clasificador que utiliza un método similar. Ya que el Random Forest no es otra cosa que muchos árboles de decisión aleatorios de baja profundidad.

El clasificador que ha obtenido los peores resultados ha sido el que utiliza el método vecinos próximos, aún qué con resultados muy similares al resto, probablemente con un

estudio en profundidad del número de vecinos óptimo podría mejorarse el resultado de este clasificador.

Haber utilizado el método de undersampling nos proporciona una desventaja frente a la clasificación directa de los datos y frente al método de oversampling. Esta desventaja consiste en que la cantidad de datos a clasificar se ve notablemente reducida con el objetivo de utilizar el mismo número de datos de cada clase, provoca que el porcentaje de acierto y error sea menos preciso. Sin embargo, nos proporciona un resultado muy fiable, puesto que cada uno de los datos y sus valores ha sido sacado directamente de los datos que ha generado el smartwatch.

5.3 Ejecución con oversampling

En la ejecución utilizando el método de oversampling, el total de datos predichos es de 11050. Al utilizar 5 particiones, cada partición clasifica 2210 datos, que han sido clasificados mediante el entrenamiento de los otros 80.

5.3.1 Naive Bayes

Tabla de resultados:

Partición	Acierto	Fallo	Acierto (%)	Fallo (%)
0	1719	491	77,78	22,22
1	1805	405	81,67	18,33
2	1797	413	81,31	18,69
3	1791	419	81,04	18,96
4	1795	415	81,22	18,78
Total	8907	2143		
Medio	1781,4	428,6	80,61	19,39

Matriz de confusión:

P/O	E	R
E	3543	1998
R	89	5436

5.3.2 KNN

2 VECINOS

Tabla de resultados:

Partición	Acierto	Fallo	Acierto (%)	Fallo (%)
0	2188	22	99,00	1,00
1	2187	23	98,96	1,04
2	2193	17	99,23	0,77
3	2191	19	99,14	0,86
4	2177	33	98,51	1,49
Total	10936	114		
Medio	2187,2	22,8	98,97	1,03

Matriz de confusión:

P/O	E	R
E	5435	90
R	24	5501

5 VECINOS

Tabla de resultados:

Partición	Acierto	Fallo	Acierto (%)	Fallo (%)
0	2170	40	98,19	1,81
1	2172	38	98,28	1,72
2	2175	35	98,42	1,58
3	2174	36	98,37	1,63
4	2160	50	97,74	2,26
Total	10851	199		
Medio	2170,2	39,8	98,20	1,80

Matriz de confusión:

P/O	E	R
E	5348	177
R	22	5503

7 VECINOS

Tabla de resultados:

Partición	Acierto	Fallo	Acierto (%)	Fallo (%)
0	2166	44	98,01	1,99
1	2162	48	97,83	2,17
2	2160	50	97,74	2,26
3	2151	59	97,33	2,67
4	2172	38	98,28	1,72
Total	10811	239		
Medio	2162,2	47,8	97,84	2,16

Matriz de confusión:

P/O	E	R
E	5308	217
R	22	5503

5.3.3 Regresión Logística

Tabla de resultados:

Partición	Acierto	Fallo	Acierto (%)	Fallo (%)
0	1761	449	79,68	20,32
1	1887	323	85,38	14,62
2	1925	285	87,10	12,90
3	1880	330	85,07	14,93
4	1895	315	85,75	14,25
Total	9348	1702		
Medio	1869,6	340,4	84,60	15,40

Matriz de confusión:

P/O	E	R
E	4516	1009
R	693	4832

5.3.4 Árbol de decisión

Tabla de resultados:

Partición	Acierto	Fallo	Acierto (%)	Fallo (%)
0	2187	23	98,96	1,04
1	2181	29	98,69	1,31
2	2190	20	99,10	0,90
3	2192	18	99,19	0,81
4	2191	19	99,14	0,86
Total	10941	109		
Medio	2188,2	21,8	99,01	0,99

Matriz de confusión:

P/O	E	R
E	5444	81
R	28	5497

5.3.5 Random Forest

Tabla de resultados:

Partición	Acierto	Fallo	Acierto (%)	Fallo (%)
0	2206	4	99,82	0,18
1	2199	11	99,50	0,50
2	2202	8	99,64	0,36
3	2202	8	99,64	0,36
4	2197	13	99,41	0,59
Total	11006	44		
Medio	2201,2	8,8	99,60	0,40

Matriz de confusión:

P/O	E	R
E	5507	18
R	26	5499

Tras examinar los resultados de las clasificaciones, podemos observar que, por muy poca diferencia con el Árbol de Decisión, el clasificador Random Forest, ha resultado ser el más preciso. Es normal que el porcentaje de error de ambos clasificadores sea similar, ya que como hemos explicado en el análisis de los modelos de clasificación utilizados, el Random Forest es un clasificador que utiliza arboles de decisión de poca profundidad para realizar la clasificación

El clasificador que ha obtenido los peores resultados ha sido el que utiliza el método de Naibe Bayes, con aproximadamente 2000 datos fallados más con que el que ha realizado la mejor clasificación.

Observando las matrices de confusión vemos que los métodos de clasificación tienden a realizar un mayor numero falsos negativos, es decir, datos que originalmente eran estados de relajación y que han sido predichos como estrés, que de falsos positivos. Sin embargo, observando la clasificación de Random Forest, ha conseguido solventar el problema, clasificando consiguiendo reducir el número de falsos negativos. Sin duda este es uno de los motivos por los que el clasificador Random Forest ha obtenido el mejor porcentaje de acierto.

Haber utilizado el método de oversampling nos proporciona el beneficio de tener un número más elevado de datos, gracias a ello, el porcentaje de acierto es más preciso que en el método de undersampling, sin embargo, hay que tener en cuenta que muchos de los datos clasificados sin datos sintéticos generados por un algoritmo de mutación, es decir, el resultado de la clasificación no está tomado de datos directamente tomados de la generación del smartwatch.

6 Conclusiones y trabajo futuro

6.1 Conclusiones

Para concluir el estudio vamos a realizar la comparación de los resultados del apartado anterior con el fin de localizar el clasificador que mayor porcentaje de acierto ha mostrado.

Undersampling		
Clasificador	**Acierto**	**Error**
Naive Bayes	87%	13%
KNN-2	86%	14%
KNN-6	87%	13%
KNN-7	85%	15%
Regresión Logística	85%	15%
Árbol de Decisión	84%	16%
Random Forest	91%	9%

Oversampling		
Clasificador	**Acierto**	**Error**
Naive Bayes	80,61%	19,39%
KNN-2	98,97%	1,03%
KNN-5	98,20%	1,80%
KNN-7	97,84%	2,16%
Regresión Logística	84,60%	15,4%
Árbol de Decisión	99,01%	0,99%
Random Forest	99,60%	0,40%

Examinando los resultados, podemos observar que, pese a que varían entre las dos ejecuciones, ambas concluyen que Random Forest, el clasificador que utiliza árboles de decisión aleatorios para predecir el resultado es el más preciso en este caso.

Esta variación entre ejecuciones se debe a que el método que utiliza oversampling utiliza un conjunto de entrenamiento mucho mayor, y por lo tanto mas preciso que el de undersampling, ya que éste ha tenido que sacrificar gran parte de los datos para mantener una equivalencia entre el numero de valores de las clases.

6.2 Trabajo futuro

Puede dividirse este apartado en distintos apartados, dado que hay mejoras de distintos tipos.

La primera de ella es la mejora del estudio. En este momento, el estudio recoge los modelos de clasificación óptimos para lo que la aplicación Taimun-Watch necesita y soporta hoy en día, esta aplicación, a pesar de estar en funcionamiento, continúa con actualizaciones de mejora muy interesante que aparecen anualmente. Por ello, es probable que en un futuro se puedan añadir clasificadores de mayor envergadura que en este caso han sido omitidos, como la red neuronal, o los clasificadores genéticos.

Otra es la integración con la aplicación. Al ser esto un estudio sobre el rendimiento de los algoritmos de clasificación en la predicción de emociones, el siguiente paso claro es la decisión e integración del algoritmo en un módulo del proyecto en Android de Taimun-Watch. Aun que lo más lógico sea la integración de un clasificador Random Forest en la aplicación, es decisión del programador de Taimun-Watch escoger el clasificador. Podría existir la posibilidad de escoger un modelo de clasificación distinto, sacrificando algo de rendimiento o de precisión en la clasificación, si se considera necesario para esta integración.

Referencias

[1] A Trainable Spaced Repetition Model for Language Learning, by Duolingo http://making.duolingo.com/how-we-learn-how-you-learn

[2] Prediction of Preterm Deliveries from EHG Signals Using Machine Learning, by Paul Fergus, October 28, 2013. http://journals.plos.org/plosone/article?id=10.1371/journal.pone.0077154

[3] How to Control a Robotic Arm with Your Mind, by Using Machine Learning, 22 Jan 2017, by Kimberley Mok https://thenewstack.io/control-robotic-arm-mind-using-machine-learning/

[4] Sentiment Analysis using Machine Learning, May 2017, by Ritika Dhania, Yogesh Ahlawat http://ijesc.org/upload/ab527f83abeac3f8e4b7651967dc89aa.Sentiment%20Analysis%20using%20Machine%20Learning.pdf

[5] Clasificación de patrones: Métodos supervisados, abril 2008, por Jordi Porta Zamorano, EPS UAM. http://www.iula.upf.edu/materials/050418porta_4.pdf

[6] Best Programming Language for Machine Learning, by Jason Brownlee on May 10, 2014. https://machinelearningmastery.com/best-programming-language-for-machine-learning/

[7] Dealing with imbalanced data: Undersampling, Oversampling and proper cross-validation, by Marco Altini 17/8/2015. https://www.marcoaltini.com/blog/dealing-with-imbalanced-data-undersampling-oversampling-and-proper-cross-validation

Glosario

TEA	Trastorno del Espectro Autista
Smartwatch	Reloj inteligente
KNN	K Near Neighbour (K Vecinos Próximos)
TFG	Trabajo de Fin de Grado

Anexos

A Manual de instalación

Para realizar la instalación del proyecto seguiremos unos sencillos pasos:

A) Descarga e instalación de Python 3.5 en nuestro equipo.

B) Instalación de las librerias Scikit-Learn e Imblearn

C) Descarga de los modulos del proyecto, se recomienda aislarlos en un directorio, de nombre 'Proyect' por ejemplo.

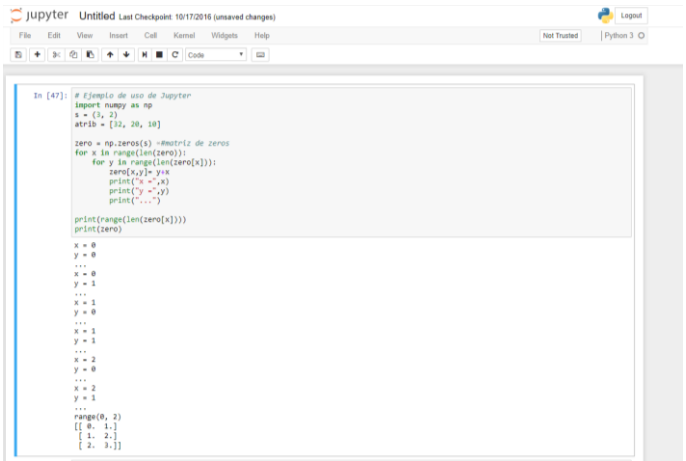
D) Descarga del fichero de datos en la ruta ../Proyect/Datos/fichero.csv

E) Ejecución del proyecto

B Manual del programador

Para modificar el código cualquier editor de texto con el que el programador se sienta cómodo es válido. Como recomendación, utilizo normalmente Atom, sin embargo, cualquier otro como Sublime Text o Notepad++ es igual de válido.

Sin embargo, para una programación que requiera mucho ajuste sobre un programa ya completo, y una representación de los resultados esquemática, he utilizado Jupyter Notebook.
Jupiter Notebook, es un editor de texto que permite separar en módulos distintas implementaciones de código, y permite expresar los resultados de la ejecución en formato Notebook, permitiendo crear tablas, títulos en un formato similar a Latex.



Para instalar Jupyter Notebook, es necesario descargar el paquete Anaconda, y a continuación realizar la instalación de librerías de forma similar al apartado anterior, utilizando la consola de Anaconda.